Development Of an Android Mobile Phone Application for Finding Closed-Loop, Analytical Solutions to Dense Linear, Algebraic Equations for The Purpose of Mathematical Modelling in Healthcare and Neuroscience Research

Santhosh Kumar Rajamani[1*], Radha Srinivasan Iyer[2]

Abstract

Closed-loop, analytical solution of Linear, algebraic equation containing many unknown variables are found in many mathematical modeling equations and network analysis involved in healthcare and neuroscience research. Finding unique, analytical solution for linear, algebraic equations has diverse application in many fields. Computation of unique, non-trivial solutions to large array containing several variables is a computationally overwhelming task. The paper begins by introducing the concept of network analysis in modelling for healthcare and neuroscience research. A simple example of network modelling is the logistics of delivering vaccine like COVID19 from a company to Primary health center while maintaining cold-storage of the vaccine. This is followed by explanation of the working of computationally efficient, best-practice LU Decomposition with partial pivoting algorithm to solve dense linear equations. This is followed by narration of building, testing, obfuscation, compiling and release of an Android Graphic user interface application implementing the above methods.

The final part of the paper examines the exceptional accuracy and efficiency of solving, dense matrix equations on Android Run Time machine using this approach. The calculated Poisson modelling of probability of Stochastic, Singularity event is $= \mathbf{3.26 \times 10^{-9}}\boldsymbol{p}$er floating point operation a number derived after running $\mathbf{15,625,000,000}$ floating-point operation runs. The mean execution time was 88.534 $\boldsymbol{seconds}$, for solving matrix equation $[\boldsymbol{A}]\boldsymbol{with}$N=60 variables in performing $\boldsymbol{N^3}$=216000 computations. The whole working Android application containing many other tools is hosted on the GitHub and Figshare platforms along with additional graphs, dataset, Java, and Python programs used to complete this study.

KeyWords:LU Decomposition, Matrix algebra, Mathematical modelling, Computer aided algebra, Linear algebra, Linear equations

DOI Number: 10.14704/nq.2022.20.8.NQ44521 **NeuroQuantology2022; 20(8):4959-4973**

Introduction

First degree equations of two to any number of unknown variables, all of which have an exponent of 1 (x1 so the highest degree is 1) are called Linear systems of equations. The word “Linear” here may remind the reader that these equations are represented by straight lines (or planes) in orthogonal coordinate systems.

There are numerous situations in science which require analytical, solutions a system of Linear equations.There are numerous instances in diverse fields like Ecology, Plant taxonomy, Business mathematics, Biostatistics, Precalculus, Electronics: Kirchhoff 's current law, Physics:

Corresponding author: Santhosh Kumar Rajamani
Address: [1]Professor, MAEER MIT Pune’s MIMER Medical College and DR. BSTR Hospital, Talegaon D, Pune, Maharashtra, India 410507 , [2] Student of M.P.Th.(Neurosciences), Dr DY Patil College of Physiotherapy, Pune, Maharashtra, IN
E-mail: Minerva.santh@gmail.com, drsanthosh.rajamani@mitmimer.com

Simple Mass-Spring Systems, Linear Programming, linear econometric models, mathematical modelling, solutions of elliptic partial differential equations and so on. In addition, even many non-linear systems can be modelled with a reductionist approach to system of Linear equations, for example Numerical or approximate solutions of partial Differential equations. They also appear in digital image processing and compression, digital signal processing, digital noise filtering in systems and operational theory of analysis as stated by Yang, M.-H., Yang, F.-Y. and Oyang, Y.-J. (2013).

Linear equations and Network analysis in Healthcare and neuroscience research

Finding unique solutions to Linear, algebraic equations are a recurring theme in many mathematical modelling and network analysis problems in healthcare and neuroscience research. For example, logistics supply chain model of COVID19 vaccination is modelled in figure1. The supply chain and warehousing of the many vaccines including Coronavirus is difficult problem, as needs continuous maintenance of cold storage temperature in the warehouse called the Cold chain. Coronavirus vaccines like the Indian Covaxin and Covishield need to be stored 2-8 degrees, and by contrast Pfizer's vaccine needs to be stored at -70°C. Managing logistics of "cold chain" is crucial to success of vaccination campaign. This simple example can be network modelled using a rooted, directed tree graph as shown below in figure1. This graph was created using free Graphing software Gephi as stated by Bastian, M., Heymann, S. &Jacomy, M., (2009). This is also a directed acyclic graph (DAC) where the edges are directed but there is no rotation.

$$Graph G_{VaccineLogistics} = (V, E)\ N_v = |V| = 7, N_E = |E| = 8$$

In the Figure1. simple example biological network Graph of order (nodes) 7 and Size (edges) 8. This network is defined by nodes or vertices are number vaccine available at each center utilized or wasted. The vaccine availability and flow are independent of the nodes. Since the chance of breaking the cold chain increases with increasing number of nodes (by repeated storage and mobilization in and out of freezer of for dispensing) the shortest walk or path through this example tree called the Geodesic would be the optimum logistical solution. In contrast, the longest path called Diameter would be the most logistical critical path in this graph. Such a healthcare network can be mathematically modelled with the following sparse matrix equation, called the Adjacency matrix which is basically the summary of fundamental connectivity within the network as described by Kolaczyk, E.D. (2010).

Adjacency matrix of $[A]a_{i,j} = \begin{cases} 1\ if i,j \in (belongsto)\ Edgeset \\ 1 \\ 0 \\ 0\ if i,j \notin (doesnotbelongthe)\ Edgeset \end{cases}$

Another useful matrix is Incidence matrix which contains directional information about the edges of the network. Incidence matrix is defined for a directed graph as following, based on paper of Seshu S. and Balabanian, N. (1967).

Incidence matrix$[A]a_{i,j} = \begin{cases} 1\ if i,j \in (belong) Edge and 1\ if arrow away from vertex a \\ -1 \quad or\ -1\ if arrow directed towards vertex a \\ 0 \quad if i,j \notin (notbelong) Edgeset \end{cases}$

Thus, the Incidence matrix $[In]$ is a square matrix, computed for graph $[A]$ like this based on the above rule.

$$\begin{array}{c} \\ \boldsymbol{Node1} \\ \boldsymbol{Node2} \\ \boldsymbol{Node3a} \\ \boldsymbol{Node4a} \\ \boldsymbol{Node3b} \\ \boldsymbol{Node4b} \\ \boldsymbol{Node5b} \end{array} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{-1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{+1} & \boldsymbol{+1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{+1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{+1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix}$$

$$\begin{array}{c} \\ \boldsymbol{Node1} \\ \boldsymbol{Node2} \\ \boldsymbol{Node3a} \\ \boldsymbol{Node4a} \\ \boldsymbol{Node3b} \\ \boldsymbol{Node4b} \\ \boldsymbol{Node5b} \end{array} \begin{bmatrix} \boldsymbol{1} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{-1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{+1} & \boldsymbol{+1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{+1} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{+1} & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{0} & \boldsymbol{-1} & \boldsymbol{0} & \boldsymbol{0} \end{bmatrix} \begin{bmatrix} \boldsymbol{x_{1(t)}} \\ \boldsymbol{x_{2(t)}} \\ \boldsymbol{x_{3(t)}} \\ \boldsymbol{x_{4(t)}} \\ \boldsymbol{x_{5(t)}} \\ \boldsymbol{x_{6(t)}} \\ \boldsymbol{x_{7(t)}} \end{bmatrix}$$

$$= \begin{bmatrix} \boldsymbol{c1} \\ \boldsymbol{c2} \\ \boldsymbol{c3} \\ \boldsymbol{c4} \\ \boldsymbol{c5} \\ \boldsymbol{c6} \\ \boldsymbol{c7} \end{bmatrix}$$

$$\boldsymbol{ormatrix, condensedform}[\boldsymbol{A}] \times [\boldsymbol{X}] = [\boldsymbol{C}]$$

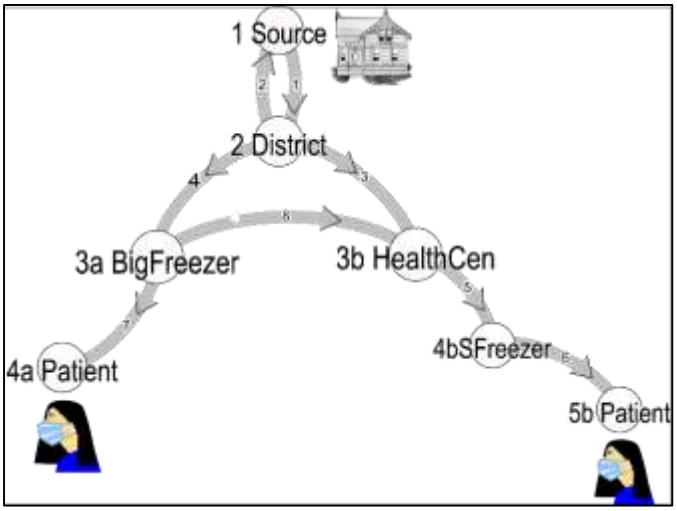

Fig1. A directed, rooted tree graph showing simple network analysis of logistics in distribution of vaccines like Covid-19 from a district hospital node(2), sourced from production, the root node (edge 1) and returned to root node (edge 2) if unused, node(3a) is cold-storage district hospital (district walk-in freezers), and node (4a) are people vaccinated directly in the hospital and node are those vaccinated in the community (5b) after storage node (4b) Small freezers in the local Primary Health centers node (3b). Node. Primary Health centers occasionally gets stocks from district walk in freezers (edge 8). This healthcare situation can be mathematically modelled using Network analysis and Matrix equations, this is based on Khanna, V.V., Chadaga. K. et. al. (2022) and Allen, L.J.S. (2007)

The first row is given by $+x_{1(t)} - x_{2(t)} + 0x_{3(t)} + 0x_{4(t)} + 0x_{5(t)} + 0x_{6(t)} = c1$ so on leading to N=8 linear equations which must be analytically solved by Gaussian elimination or any other methods, and unique values of $x_{1(t),}x_{2(t),}$ calculated in a method, as described by Ma'ayan A and Macarthur, B.D (2012).. This situation being aided by sparse nature of the matrix and the nodal relations are not so simple in many real-life situations. With the increasing number of variables, the number of steps required to manually solve this system of equation by for example Gauss Elimination (by reducing the matrix to reduced row echelon form by series of elementary row transformations, as detailed by Yang, W., Zhang, D. et.al. (2021)) becomes very large. Most of the manually solved examples in textbooks contain real numbers from 1 to 10, which are hardly ever found in real life equations.

Objectives

Analytically solve dense linear equations of size (N) limited only by Operating system limitation or RAM on Android cell phones using LU Decomposition and Partial pivoting in Android Java language (Android studio).

Methodology

Generally, the unknown variables in the System of Linear systems of equations are represented by x (same as x, y, z or 3 variables in standard books). We designate a square matrix $[A]$ containing N unknows of size N^2 where $N = \{ n \in \mathbb{N} \; and n > 2\}$
Variables/Unknowns
$x_1(x), x_2(y), x_3(z), x_4, x_5 \ldots . x_N$ where n can be any natural number (but not zero)
$\vec{X} = \left(\overrightarrow{\langle x_1 \rangle}, \overrightarrow{\langle x_2 \rangle}, \overrightarrow{\langle x_3 \rangle}, \overrightarrow{\langle x_4 \rangle} \cdots \overrightarrow{\langle x_N \rangle}\right)$
Column matrix $[X]$ is also called a vector of dimension or axis one and variables on this axis are called the components of this vector. The number of N is called the dimension of the vector space as stated by Abadir, K.M. and Magnus, J.R. in their paper (2005).
Common equations are thus written in this convenient form for the purpose of this paper and computer program.

$ax + by + dz = k$ which is synonymous with $a_{11}x_1 + a_{12}x_{12} + a_{13}x_3 = c_1$ so on for ith equation thus ai1 + ai2 +.... or Summation of $\sum_{i=1}^{N} a_{ij}x_i = c_i$. The term Cn is a column matrix (1 column only) also called a vector C thus $\overrightarrow{[C]}$ and column matrix $\overrightarrow{[X]}$, the equation is thus

represented by vector dot product $\overrightarrow{[X]} \cdot [A] = \overrightarrow{[C]}$. This is called the Dot product Vector representation of a linear equation (as described at this site http://www.wolframalpha.com (1996).

Matrix Representation of an equation is the depiction of a linear equation as a product of a square matrix with 2 column matrices as stated by Stewart. G.W. in his book (2000).

$$[A] = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{13} \\ a_{23} & a_{32} & a_{33} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \end{bmatrix}$$

So, on…

$$\begin{bmatrix} a_{i1} & a_{i2} & a_{i3} \dots . a_{ij} \end{bmatrix} [x_i] = [c_i]$$

Or simply

$$[A] \cdot [X] = [C]$$

Simply rearranged

$$[X] = inverse([A]) \times [X] = [A]^{-1} \times [C]$$

$$= \frac{adj[A]}{det|A|} \times [C]$$

On the precondition that $[A]$ is a square non-singular matrix.

For the sake of simplicity, the above notation is going to be used throughout this paper, with arrows omitted and is a synonymous with any other representation as stated by Xian-Da Zhang (2017).

Another representation is Augmented matrix representation, this is not used as this is notation for hand solution by Gaussian elimination by reducing to row echelon form by performing hand, elementary row transformation on both sides of the matrix.

$$[A] = \begin{bmatrix} x_1 a_{11} & x_1 a_{12} & x_1 a_{13} \mid c_1 \\ x_2 a_{21} & x_2 a_{22} & x_2 a_{13} \mid c_2 \\ x_3 a_{23} & x_3 a_{32} & x_3 a_{33} \mid c_3 \end{bmatrix}$$

Diagonal Matrix is one in which all elements below the diagonal line are zero, but diagonal line itself is nonzero. An Identity matrix has 1 in the diagonal line.

$$diag[A] = \begin{bmatrix} a_{11} & 0 & 0 \\ 0 & a_{22} & 0 \\ 0 & 0 & a_{33} \end{bmatrix} IdentityJ = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A permutation matrix P is a binary matrix (of 1, 0 only) which has 1 in each row and each column. This matrix (as we will see further) is an Operator with causes interchange of rows, like this example.

$$\begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix} \times \begin{bmatrix} a & b \\ c & d \end{bmatrix} = \begin{bmatrix} c & d \\ a & b \end{bmatrix}$$

A sparse matrix is a type of matrix with sufficient zeros so that they can be manipulated in a computationally efficient manner. In contrast a dense matrix is neither is repetitive nor sparse to help in reaching a unique solution. The objective is to solve this dense system of linear equation in finite number of steps in a most efferent way called closed-loop solutions as described by Debonis, M.J. (2022). Open loop or trivial solution are of not much interest like for example $x_1 = 0, x_2 = 0, x_N = 0$, will solve any equation (even quadratic equation) but are not of any real utility as stated by Bronson R. and Costa, B.G. (2009).

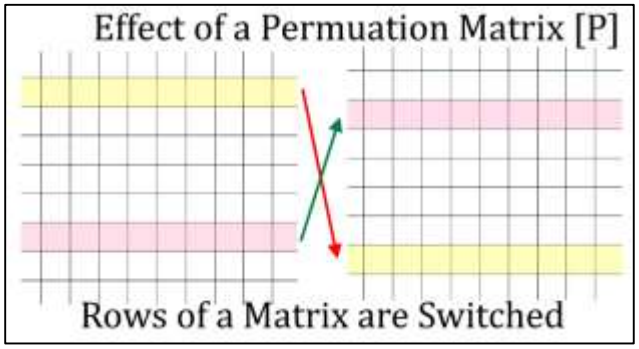

Fig.2. Schema of Partial Pivoting by the row-shifting effect of product with Permutation matrix [P]

Methods of arriving at closed-loop solution of Matrix equation $\overrightarrow{[X]} \times [A] = \overrightarrow{[C]}$

There are many ways such a dense, matrix equation maybe solved which are broadly classified into:

1. Stanimirovi, I. (2017) observed that Analytical methods or Direct methods by following a specified algorithm producing accurate solutions to $[X] \times [A] = [C]$, these are useful for dense, non-sequential matrix equations, examples are Cramer's rule, Inversion of Matrix, Gauss Jordan method of direct elimination by inspection, and LU Decomposition.

2. Young, D.M. (2003) stated that Numerical methods or Iterative methods by sequence of successively better and better approximations of the solutions, these are useful for sparse,

sequential, or structural matrix like Toeplitz's, Hankel's, Vandermonde, Cauchy, Pick-ups, Bzout's and Toeplitz' matrices. To reminisce a few extant numerical methods: Gauss-Seidel method, SOR, SSOR, Richardson's method, and Jacobi method.

Problem with common classroom methods of solving large Linear equations

Golub, G.H., Chan, R.H., Greif, C., O'Leary, D.P. and Ebrary, I. in their paper (2007) observed that for example, solution of Matrix equation of order n using rearranged $[X] = inverse([A])[C]$, computation of $[A]^{-1}$ of a square matrix of size N will need to solve (N+1) number of determinants and perform$N + 1$! factorial number of operations in terms of multiplications and additions. For example, solving a linear equation having 6 terms will need 7! number of operations which will 5040 accurate steps by hand. Analytical, closed-loop solution of linear equation by Cramer's rule with 10 terms will need a staggering 3,628,800 steps.

Lower and Upper Triangular matrix decomposition

LU decomposition is attributed Alan Turing in 1948, who showed the factorization of a Matrix into 2 triangular, Matrices: The Upper $[U]$ and Lower $[L]$matrix is unique (proof is given in all textbooks) for any square, non-singular matrix. LU factorization is the computationally most efficient algorithm for solving such a system of linear equations, in contrast to other methods in literature like Block LU decomposition, Rank factorization, Cholesky decomposition, QR decomposition, and singular value decomposition. For instance, Cholesky decomposition produces contain square roots which are computationally difficult to solve and QR decomposition method $^4/_3 N^3$ takes twice the number of computations to achieve the same results as observed by AnatolijDmitrievicMyskis (1972).

LU Decomposition/ factorization is a more efficient implementation of Gauss direct elimination method. This efficiency allows for solving Matrix equations on a mobile processor, a task which was relegated to Supercomputers in the past. This is factorization scheme is shown in picture that follows.

Any Square Matrix as factors of 2 Square matrices. Such an operation is possible provided matrix is non-singular.

$[A] = [L] \times [U]$

Such a factorization is particularly beneficial in solving revolving differential equations whose $\overrightarrow{[C]}$ direction of Vector changes due to revolutions of the vector.

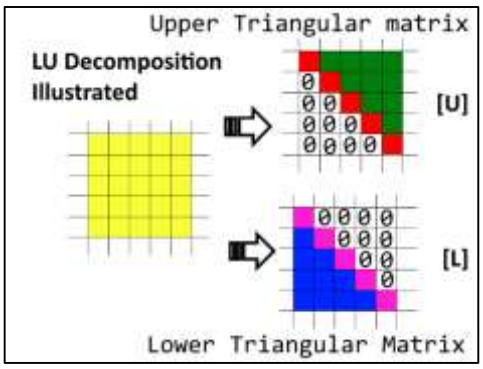

Fig.3. Schema of Factorizing a matrix [A] into an Upper and Lower triangular matrix for purpose of fast and efficient computation

Special properties of diagonal elements of L and U matrices making computations easy

Computing determinants is easy for Upper and lower Triangular matrices and involves Product of the Diagonal elements of the matrices.

$$|A| = \begin{bmatrix} a_{11} & 2 & 4 \\ 0 & a_{22} & 7 \\ 0 & 0 & a_{23} \end{bmatrix} = a_{11} \times a_{22} \times a_{33} \times a_{44} \ldots \times a_{ii}$$

Similarly inverse of Lower Triangular matrix is simply the sign inversed elements below the diagonal.

$$[L]^{-1} = (-1) \times [L]$$

Solution of $[X]$ is easily reduced to finding inverse of $[U]^{-1}$ and matrix multiplications.

$$[X] = [A]^{-1} \times [C] = [L]^{-1} \times [U]^{-1} \times [C]$$

LU Decomposition elegantly, simplifies the closed-loop solutions of complex, system of equations. So, the Upper and lower Triangular matrices can be easily used to find several other closed-loop solutions like Determinant of$|A|$, Inverse of matrix $[A]^{-1}$so on. This flexibility, ease and computational efficiency makes the LU factorization the preferred analytical method of solving matrix equations as described by A. G. Kurosch (1988).

Smaller number of floating-point operations (fops) or steps making computations easy

The number of floating-point operations (fops) or steps need to complete LU factorization of a dense, Linear equation of order N is given by the formula $\mathcal{O}(\mathcal{N}^3)$.The number of floating-point operations (fops) needed to solve a matrix of size N is given by the equation $\mathcal{O}(\mathcal{N}^3)$when $\mathcal{N} \in$ can be any Natural number $\mathcal{N} > 2$. and $\mathcal{N} - 1$ division operations, which are computationally complex. Therefore, an equation with 6 unknowns will need only 216 steps and equation with 10 unknowns can be solved in 100 accurate steps, for 20 it is 4000 steps only as stated in paper by Bisseling, R.H. (2020). In contrast, the LU factorization process of large numerical arrays is not without problems as we shall see in next section.

Computational problems inherent to factorization of a Matrix into U and L Matrices

Non-Existence of a Lower and Upper Triangular matrix

The LU-decomposition is computational efficient, best-case algorithm but this method may fail in some, peculiar matrices.

Problem of Singulate or degenerate Matrix and Singularities in calculation

A matrix $[A]$ is non-singular on the condition that there exists such a matrix

$$[A] \times [A]^{-1} = [A]^{-1} \times [A] = [I]$$

$[I]$ is the identity matrix, element 1 along the diagonal and rest all the non-diagonal elements are zero. A singular matrix is also called degenerate matrix and determinant is zero $|A| = 0$, such a system of matrix cannot be consistent giving no solution or infinite solutions (open-loop solutions) cases. Thus, singularity of matrices occurring both naturally and those arising during computations by truncation of numbers to a finite number of digits, are a hurdle towards a successful analytical solution.

A singular or degenerate matrix can appear to be non-singular on cursory examination. The prechecking by computation of $determinant|A| \neq 0$ can immediately resolve the matter by as we have seen (section on Cramer's rule) this is computationally an inefficient method of solving matrices as observed by Proskuryakov, I.V. (2011). A matrix is $[A]$is non-singular on the condition that the diagonal elements are nonzero stated by Golub, G. H., Van Loan, C. F. (1996). A condition which can be easily checked by diagonal element of $a_{i,j} \in [A]$ i (row) = j (column) and $a_{i,j} \neq 0$ condition in loop. This cursory evaluation will later fail as singularity may arise due to loss of precision especially while rounding errors due to truncation of digits or bits in floating point operation, such matrices are said to be ill-conditioned. Prechecking by finding the Determinant of a large array is wasteful and not efficient for large matrices as described by Xian-Da Zhang (2017). Since definition of limit of singularity in calculating matrices is defined $\lim_{ai,j} a \to 0$, we may immediately note that the magnitude of the determinant is the simple quantification of an ill-conditioned a matrix.

$$\lim_{n} |A| \to 0$$

A more precise, Perturbation theory-based formula for condition of a matrix (ill-behaved, singularity) is the computation of formula described by Sewell, G. (2017).

$$conditionof[A] = \|A\|\|A^{-1}\| \to 1$$

This loops us back to the paradox of Matrix LU decomposition, which is illustrated in flowchart Fig.3.

A simple example of ill-conditioned matrix which produce singularity in computations, benign looking Hilbertian matrix as demarcated by Costa, R. (2020). Defined by this equation $Hilbertianmatrices[H] = \frac{1}{(i+j-1)} fori, j =$ 1,2,3,4 $soonNforlargeN$, a smaller example is $\begin{bmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$ and $\lim_{n} |H| \to 0$ (Sewell, G. 2017).

Lotkin's matrices which are identical to the Hilbertian matrix, first row is completed with 1 only (Sewell, G. 2017).

$$[L] = \begin{bmatrix} 1 & 1 & 1 \\ 2 & 3 & 4 \\ 3 & 4 & 5 \end{bmatrix}$$

Using floating-point arithmetic of computer system when an operation of summation of n numbers is performed, each number contributing to an infinitesimally small number $\delta d_1 \cdots \delta d_n$ computed the exact sum of the same n numbers perturbed by

infinitesimally relative errors. The errors add up in computation like this.

$$\sum X_n(Computedwitherrors) = X_1(1+\delta d_1)+\ldots+X_n(1+\delta d_n)$$

Loss of Precision

Since floating point operations on computer processor are only performed to a limited precision the computed analytical closed-loop solution differs from the real solution obtained by hand. This happens because $\lim_{n\to 0} 10^{-n} \approx 0 \therefore$ Matrix has become Singular / degenerate and solutions have become open loop in midst of computation.

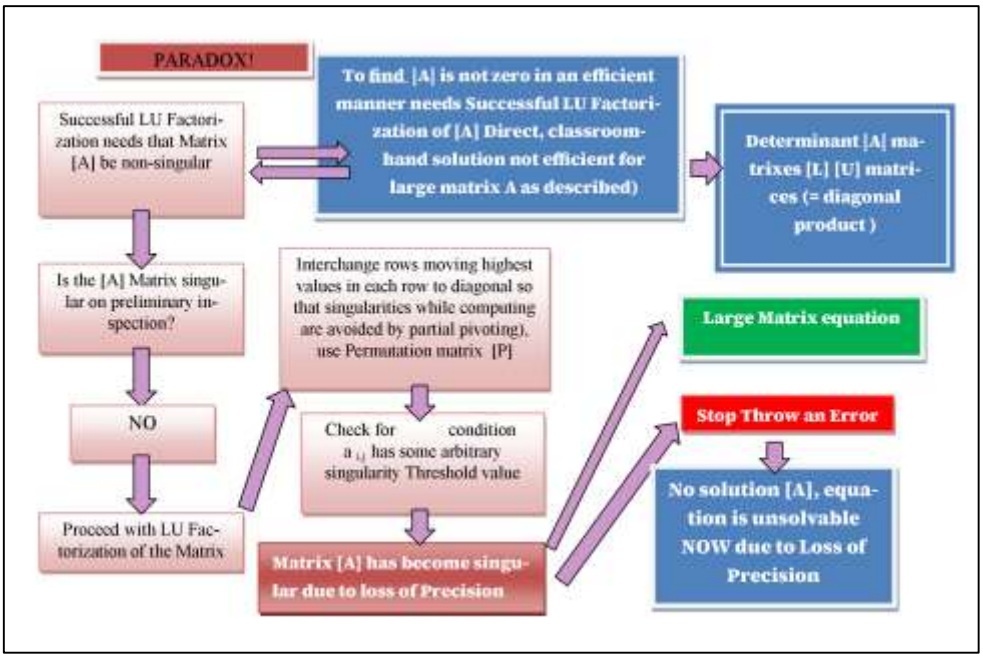

Fig4. The algorithm of avoiding Singular matrices in large matrix computations and the paradox of LU factorization and solving the Determinant of a large matrix. This paradox is narrated thus, the matrix can be factorized, efficiently if determinant|A| $\neq 0$ but to find the determinant|A| we need to factorize the matrix successfully and find L and U matrices (Xian-Da Zhang 2017). The singularity threshold of Apache Java Math library (commons-math 3-3.6) is arbitrarily set to singularity threshold $10^{-11} \approx 0$ is effectively zero, this I tested on Mobile CPU, ART Android Runtime (above KitKat) and found that actual truncation errors began at orders $\gg 10^{-308}$ only but this might be a feature of the ART machine of the benchmarked tested mobile.

LU factorization by partial pivoting using a Permutation Matrix $[P]$

Pivot is the first and the left most nonzero element in a row of a matrix, beginning at which is the starting point of elimination by Gauss method. We find that the first pivot element a1,1 =0 or, $\lim_{a} a_{1,1} \approx 0$, we need to Pivot or rearrange the rows so that the row $a_{p,1} \neq 0$, this process is called Partial Pivoting sated by Kaw, A.K. (2008).

Interchange of both rows and columns is complete pivoting. Complete pivoting is more reliable in avoidance of matrix singularity, involves testing each element in row and column. Conversely, this places considerable computational overhead, destroys the structure of a matrix and cannot be employed on blocked matrices and so is not considered an ideal method stated by Sewell, G. (2017).

Partial Pivoting for size is agreed best tradeoff method in terms of accuracy, algorithm stability, computational efficiency works well real-life situations. The concept of pivoting is simple, but it is not computationally trivial matter to decide which element to use as pivot. A row is searched, and number of largest magnitudes, i.e., modulus in case negative number is found and moved to the location (1,1) in the matrix. This step is repeated successfully in each row as described by Malyshev, A. N. (1995) and Golub, G.H.; Meurant G.A. (2006). Other pivoting strategies are Scaled pivoting by calculating ratios and Rook Pivoting.

The existence of such a row is attested by that fact that determinant of [A], $|A| \neq 0$. This is possible by left product of A by a Permutation matrix $[P]$, whose properties we already summarized. The proposed analytical, solution for the Linear equation of N size is as follows.

$[P] \times [A] = [L] \times [U]$

Permutation matrix is a simple binary matrix (of only a=1 and 0s) to allow switching of rows, and please note that inverse matrix of $[P]^{-1}$is same as [P].

$[P] = inverseof[P]$

Or

$[A] = [L] \times [U]$

Analytical closed-loop Solutions of system of Linear equations using LU factorization is by following steps

$[A] \times [X] = [C]$

$[L] \times [U] \times [X] = [P]^{-1} \times [C] = [P] \times [C]$

Substitute a column matrix $[Y] = [U] \times [X]$

Then forward solve, called forward sweep, substituting $[L] \times [Y] = [P] \times [C]$ for values in matrix $[Y]$

Then backward solve, called backward sweep substituting matrix $[Y]$ in $[U] \times [X] = [Y]$ or finally

$[X] = [U]^{-1} \times [Y]$

The forward sweep places an overhead of $\mathcal{N}(\mathcal{N} - 1)$ floating-point operations and backward sweep places $\mathcal{N}(\mathcal{N} - 1) + \mathcal{N}$ floating point operations, compared this to the actual LU decomposition step which is $O(\mathcal{N}^3)$ as clarified by Meurant G.A. (2006)

Choice of the available library Apache Math Common

Apache Math Common (commons-math3 version 3.6) is a Java language library of lightweight, self-contained, common mathematical and statistics libraries for solving common computer science problems, available from the website https://commons.apache.org/proper/commons-math/.

Most of the algorithms used in this package are optimized and best practice to utilize least amount of system resources in Computation and provide reliable results on desktop and server versions of Java. We could not find any previous reference or experience to porting this desktop Java library to the ART Android Run Time Processor (C.P.U).

Issues in actual implementation of the LU Decomposition

I faced the following issues / problems in actual implementation of the algorithm

Limited memory RAM and CPU computation power of a cell phone to perform Matrix computations (Supercomputers were used)

Create an error free GUI interface (also use a lot of 9-patches as background for Input dialogs)

Validation of user input

Check is the Matrix is square by checking number of row equal to each column of the Matrix (cannot solve a ragged 2D array)

Check is the system of linear equations has an analytical, closed-loop solution (reached within finite number of steps)

Check is the Matrix is singular on cursory evaluation

Feasibility of porting the Apache Java library to Android (described here first time there is no documentation extant)

Choosing Partial pivoting row to avoid singularity of matrix in midst of calculations due to retention only a finite number of digits (or bits) in floating point calculation using singularity threshold $10^{-11} \approx 0$ is effectively zero.

Floating point operation without leaking memory

Accurate output

Complete Android Test and Lint error free, push App to debug stage

Use Object Oriented programming (no go to jumps!) and reuse cases

Deal with issue of obfuscation of Java classes by ProGuard (dealt with successfully for first time in this paper), push Java compilation to App release stage

Successful error free compilation on Android studio

Document LU factorization in terms of speed and accuracy

Modelling Matrix singularities due to truncation in Android Run Time using Poisson Distribution

Poisson distribution is used in analysis of events which are sporadic and have a rare likelihood of occurrence. A simple rule of thumb for choosing Poisson distribution as a decent approximation to Binomial distribution if N (number of variables or computations both criteria use in this study) is larger than $\gg 20$ and $N \times P \ll 10\, oreven < 1$ which is true in our study. Poisson distribution is a limiting case of Gaussian distribution when the probability of a rare event P, $\mathrm{Lim}\, P \to 0$, is diminishingly small and the $LimQ \to 1$, Q is large, limiting to 1, where is P is probability of rare event

happening and Q probability of event not occurring. A specimen of rare event P is Truncation errors leading to singularities despite using partial pivoting procedure. Poisson probability of obtaining N positive outcomes is given by

$$P(N_{rareEventErrors}) = \frac{\mu^N \times e^{-\mu}}{N!}$$

Where μ is population mean or also same as variance related this way $\sigma^2 = N \times P \times QandLimQ \rightarrow 1, \text{Lim}\, P \rightarrow 0$ then we can deduce that $\sigma^2 = N \times P \approx \mu$.

Materials

Graphical User Interface development (G.U.I) on Android Studio development process

The program for implementation of LU Decomposition with partial pivoting was done on Android Java version 8, Android Studio version Gradle Version 1.5.0 and JetBrains Editor. Release compilation was done targeting Android Software development kit Version 23.0, though application runs fine on any Android mobile phone version above 9 (Google Developers, 2019). This was achieved by including appcompat library version v7-23.1.1, to ensure backward compatibility with older versions of Android phones (Android Developers, 2021).

ProGuard is a Java optimizer for Android apps which makes execution 20% faster and compresses app up to 90%. ProGuard also protects against decompilation and reverse engineering of the application. ProGuard works for both Java and Kotlin. ProGuard's Gradle plugin was applied via the command in the file build.gradle, using command, apply 'plugin: com.guardsquare.proguard'. ProGuard version was applied to obfuscate the classes which caused the loss of classes leading to crashing the app on beta testing. To avoid obfuscation of essential classes in Math 3, Linear Algebra library for successful operation, the following lines were added to the text file which contains customized rules for application of ProGuard, proguard-rules.pro (www.guardsquare.com, n.d.).

-keep class org. apache. commons. math3.linear. **
-keep interface org. apache. commons. math3.linear. **
-dontwarn org. apache. commons. math3.geometry. **

The release version of application was also Zip Aligned by the command, zipAlignEnabled true and Gradle command 'gradlewassembleRelease' was used to compile the release candidate application. (www.guardsquare.com, n.d.)

A 9-Patch image is a background image which can dynamically accommodate with contents of a view like text or image without pixelating or blurring. 9-patches are very useful in highlighting the Edit text/ Input fields for user input. 9-pathches were used extensively for building the GUI of the application. These are capable to expanding to accommodate increasing amounts of numerical input. left and top are for scaling, and the right and bottom are for content. The left and top margins of a rectangular image will stretch to fit, and the contents will be shown padded by the black lines along the right and bottom margins. Android studio provides a Java Standalone desktop application for creating and editing 9-patches.

Android Application was successfully complied, and release version produced and uploaded to the GitHub.com site available at also at online research repository Figshare.com as for general repository, links are available in "Data availability statement" section at end of this paper.

Operation of the Graphical User Interface development (G.U.I)

The G.U.I is an Android drawer layout (Android Library recyclerview-v7-23.1.1) with a floating action button titled "RUN". The matrices are input by the user using the soft keyboard that pops up on touching the green area which is the Matrix $[A]$. The Matrix $[C]$or the Column vectors on right side of matrix equation $[\boldsymbol{A}][\boldsymbol{X}] = [\boldsymbol{C}]$ are fed on the blue area. Matrix columns elements are separated by single Space key press and rows elements are separated by an Enter (↲) key press. The linear equation of N size can then be solved by pressing the "RUN" button in right lower hand corner. The program then proceeds to convert the String array to into a 2-dimensional array or matrix. After cursory validation for Singulate matrix, the analytical solution of is depicted in the ListView along with Determinant of a N size matrix equation.

Fig5. The Graphical user interface G.U.I of the release app that computes the unique closed-loop, analytical solutions to dense linear, algebraic equations represented by matrix equation $[A][X] = [C]$, with N unknown variables by LU Decomposition with partial pivoting algorithm on an Android mobile. The blue matrix and green matrix are 9-patches which will dynamically resize to enclose the user input into the matrices. The green area is the left-hand side of the equation matrix $[A]$ and the blue area is the right hand side matrix $[C]$.Matrix $[X]$ of the unknowns $(x_1, x_2, x_3, \cdots\ x_N = \mathcal{N}\ variables)$are automatically inferred from the N= size of the rows or columns of square matrix provided by the user. Additional padding with zeros is needed if the coefficient of any linear equation is zero like so $0 \times x_1$ so that the matrix $[A]$is converted into a square with rows = columns. This app is hosted on the GitHub platform for free download and use (https://github.com/kephalian/Android-Dense-Linear-Equation-Solver).

Observations

Benchmarking and Beta testing of the Application

Since matrix computation are most resource consuming these were done on Supercomputers and Main frames few decades ago, now these same number-crunching, intensive matrix is capable of being executed on an ordinary, middle range Samsung A30 mobile phone [26].

The objective of this benchmarking was to demonstrate the robustness, validity, accuracy, and efficiency of matrix processing on the Android CPU.

1.Test the robustness of LU Decomposition algorithms with partial pivoting on a mobile processor

2. To test accuracy by computing solutions with extremely small values in matrix $[A]$ and occurrence computational singularity arising out of truncation digits and to model the outcomes on a Poisson distribution mass density function

3.To analyze the Execution speed of the program and correlating with Number of equations (N), Number of floating-point operations and Size of the matrix $[A]$ $(N \times N)$.

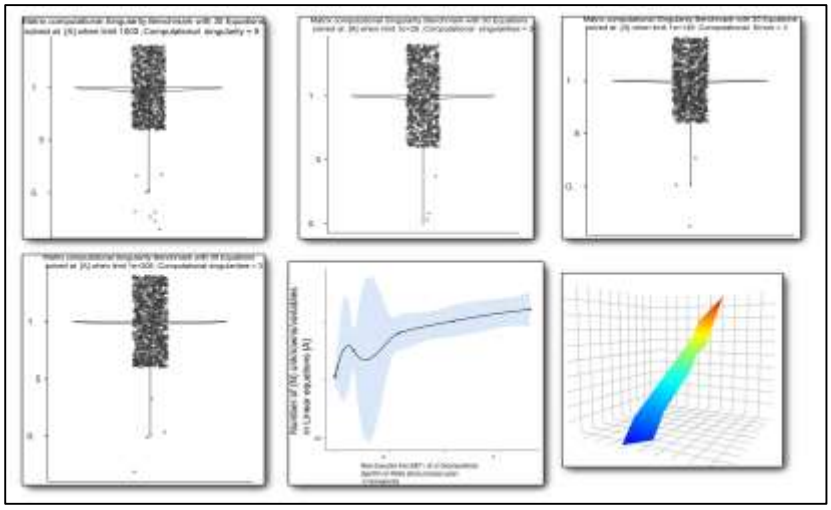

Fig6. Modelling the incidence of Singularities in midst of matrix computation by feeding progressively smaller values in the Matrix LU decomposition algorithm. The density of successful analytical closed-loop solution (Poisson variable Q→1) is plotted, and outlier Poisson event P is shown. Since the occurrence of singularity is a stochastic, discrete, time event it is modelled best by Poisson distribution. Thermal 3D surface plot of the density shows increasing events with time, depicted by upslope, and increased red colour (like increasing temperature) confirming that these errors are a type of time-dependent Poisson events.

Table1. Frequencies of Matrix singularity observed in 2 Poisson event: Poisson event 1, Q Limit →1 is successful matrix computation and Poisson event 2, Limit P→ 0 is failure to compute due to accumulation of rounding errors caused machine truncation of digits in floating-point operation. Error rate in the machine is calculated to be 51 out of 250×1000 variables = 2.04×10^{-04} per N (variable) and gross error rate per computation is calculated by $1000 \times N^3 = 1000 \times 250^3 = 15,625,000,000$ fops hence 3.264×10^{-9} per floating point operations.

Test Number	Benchmark test of computation	Incidence of error or Singularity out of 1000 runs for unique analytical solutions	Computed P value following Poisson event P of error, N^3 fops performed 1000 times
	N= 10 Equation at limit 10^{-308}	4	4×10^{-6}
	N=10 Equation at limit10^{-298}	3	3×10^{-6}
	N=10 Equation at limit 10^{-250}	6	6×10^{-6}
	N=10 Equation at limit 10^{-148}	3	3×10^{-6}
	N=30 Equation at limit 10^{-100}	9	3.33×10^{-7}
	N=30 Equation at limit 10^{-60}	3	1.11×10^{-7}
	N=30 Equation at limit 10^{-38}	5	1.852×10^{-7}
	N=30 Equation at limit 10^{-26}	3	1.11×10^{-7}
	N=30 Equation at limit 10^{-20}	4	1.48×10^{-7}
	Singularity Benchmark N=30 Equation at limit 10^{-8}	3	1.11×10^{-7}
	N=30 Equation at limit 10^{-3}	8	2.96×10^{-7}

Modelling Singular matrix incidence in computation of non-trivial solutions by inputting extremely small, randomized values in elements matrix $[\boldsymbol{A}]$		Total 11 tests X 1000 runs each X total N=250 X 1000	$\sum X_1(1+\delta d_1)+\ldots.+\delta d_n) = 51$	$P_{error}(1error) = 2.4 \times 10^{-4} pervariablecomputed,$ $P_{error}(2\ errors)\ is = 2.08 \times 10^{-8} perunknownvariableX$ $P_{error}(1\ error) = 3.26 \times 10^{-9}$Per floating-point operation $P_{error}(2\ errors)\ is = 5.34 \times 10^{-18}$ per fops

Random number generator was used to populate the matrix of $[A], and [C]$($[X]$ is inferred from size of N). The number of variables in the test were initially linear algebraic equations of 30 variables (square matrix size $[A] is N^2$=300), later reduced equations of 10 variables (Matrix size 100). Then these values were divided serially by $10^{-3}, 10^{-8}, 10^{-20}, 10, 10^{-38}, 10^{-60}, 10^{-100}, 10^{-148}, 10^{-250}, 10^{-298} and 10^{-308}$, in an attempt to stimulate computational singularity by truncation of errors. The results are shown in table below. Beyond 10^{-308} we encountered ART Android Run Time machine error due to underflow and analytical solutions are not possible on the tested model of mobile phone.

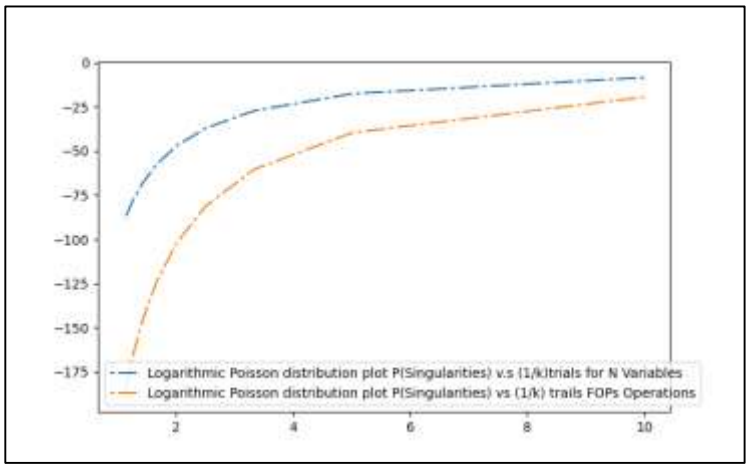

Fig7. Mathematical Modelling of the ART CPU truncation errors leading to Matrix singularity based on Poisson distribution of K trails showing asymmetrical curve, skewed towards the right, or positively skewed as probability density is nearly zero (as $\boldsymbol{P \ll Q, \mu = \sigma^2 = N \times P}$), Logarithmic probabilities P of K trails were plotted against 1/k as P is very small. This skewed curve has Kurtosis of $\frac{1}{\sqrt{\mu}} = 70.014\ and\ 17503.50$.

Modelling based on the data available $P_{error} or \mu = 2.04 \times 10^{-04}$per N (variable) and $P_{error} or \mu = 3.264 \times$ per 10^{-9}fop in Poisson model we can see that.

$$P(N_{rareEventErrors}) = \frac{\mu^N \times e^{-\mu}}{N!}$$

so $P_{error}(1\ error) = 2.04 \times 10^{-4} pervariablecomputed, P_{error}(2\ errors)\ is = 2.081 \times 10^{-8}$per unknown x solved, in other terms of Floating-point operations per operation performed on ART machine $P_{error}(1\ error) = 3.26 \times 10^{-9}$Per floating point operation performed on Android Run time machine,$P_{error}(2\ errors)\ is = 5.327 \times 10^{-18}$ per fops. Indeed, this shows the advanced computational abilities of modest, cell phone

processor and the robustness of the algorithm employed.

Analyses of the Execution speed of the program and correlating with Number of equations ($\boldsymbol{N}$), Number of floating-point operations and Size of the matrix $[\boldsymbol{A}]$ ($\boldsymbol{N} \times \boldsymbol{N}$).

The execution speeds were tested for benchmark by increasing the number of variables solved from N = 2 to 60 variables, computation N3 = 8 fops to 2,16,000 fops in a linear algebraic equation run X 1000 times (8000 fops to maximum mind-boggling 2,16,000,000 fops). The following are descriptive data regarding the benchmark tests.

Table2. Average execution times in $\mu seconds$for solving dense, linear equations of $[\boldsymbol{A}]$ on Android mobile phone Samsung A30. The number of runs was 100 and the number of variables varied from N= 2 unknowns, needing 8000 fops to N=60 unknowns needing 2,16,000,000 fops or accurate steps.

Serial No.	System of 2 Linear Equations LU Decomposition microseconds 10^{-6}	System of 4 Linear Equations LU Decomposition microseconds 10^{-6}	System of 10 Linear Equations LU Decomposition microseconds 10^{-6}
Mean Execution Time	0.0698 $\mu seconds$	0.173 $\mu seconds$	1.004 $\mu seconds$
Std. error mean	0.610	0.694	1.51
Maximum	50.6 $\mu seconds$	127 $\mu seconds$	591 $\mu seconds$
Minimum	246 $\mu seconds$	375 $\mu seconds$	1315 $\mu seconds$
	System of 40 Linear Equations LU Decomposition microseconds 10^{-6}	System of 50 Linear Equations LU Decomposition microseconds 10^{-6}	System of 60 Linear Equations LU Decomposition microseconds 10^{-6}
Mean Execution Time	29.213 $\mu seconds$	54.164 $\mu seconds$	88.524 $\mu seconds$
Std. error mean	173	431	604
Maximum	8105 $\mu seconds$	12567 $\mu seconds$	17969 $\mu seconds$
Minimum	43968 $\mu seconds$	227375 $\mu seconds$	99430 $\mu seconds$

The Karl Pearson's r correlation coefficient (r) between the variables Mean execution times and Number of variables N was calculated to be + 0.938 (*** p < .001), in addition Number of Floating-Point Operations Performed in calculation $N^3$and Number of Linear Equations with N Unknowns was calculated to be + 0.924 (*** p < .001),

As logically as expected to be a positive, linear correlation, which means that the execution time increases linearly with the increasing complexity of calculations required to solve the matrix equations.

Conclusion

This paper is a distillation of my experience in building and testing an Android application which provided closed-loop analytical solution to dense

linear equations which contain any arbitrary number of variables for the purpose of modelling and network analysis in healthcare and neuroscience research. The paper begins by introducing the concept of network analysis in modelling for healthcare and neuroscience research. A simple example of network modelling is the logistics of delivering vaccine like COVID19 vaccine from a company to Primary health center while maintaining cold storage of the vaccine. Next, a brief algebraic background on Matrix equation methods of solutions, the paper describes the paradigm of LU decomposition with partial pivoting including its triumphs and drawbacks. The implementation of LU factorization of matrices on computational machines with truncation of digits is a hurdle in successful and resource efficient factorization. The paper describes the process in development of the Android application and concerns in building and compiling the release application to Android machine code. Testing the robustness, efficiency, and reliability of this implementation on an Android Samsung mid-range mobile phone is described next. Computational singularity event or error of Matrix computations encountered in solving matrix algebraic equations are rare, independent, discrete, stochastic events and are mathematically modelled on Poisson Probability mass function. The calculated Poisson modelling of probability of Stochastic, Singularity event is $= 3.26 \times 10^{-9} per$ floating point operation a number derived after running 15,625,000,000 floating-point operation runs. The mean execution time was 88.534 $seconds$, for solving matrix equation $[A] with$N=60, 60 variables and 3600 rows and columns, performing N^3=216000 computations. As anticipated, the CPU execution times show statistically significant large, positive correlation (Karl Pearson's r correlation coefficient (r) is + 0.938, p value < .001) with increasing complexity of equations (N) and sizes of matrices (N^2).

Data Availability Statement

Android Application was successfully complied, and release version produced and uploaded to the GitHub site for free download, and available at online research repository Figshare.com (https://figshare.com/) as per Springer publications guidelines for general repository at the following site:
https://github.com/kephalian/Android-Dense-Linear-Equation-Solver
The compiled Android application, datasets, statistical analysis files, extra graphs, Java, and Python sources generated during the current study are available in the (https://figshare.com/) repository:
https://figshare.com/projects/Android_Application_Dense_Linear_Equation_Solver_using_LU_decomposition_and_partial_pivoting_for_mobile_phones/143931

References

A GKurosch (Aleksandr Gennadievich (1988). Higher Algebra. Moscow: Mir.
Abadir, K.M. and Magnus, J.R. (2005). Matrix algebra. New York: Cambridge University Press.
Allen, L.J.S. (2007). An introduction to mathematical biology. Upper Saddle River, Nj: Pearson; Prentice Hall.
AnatolijDmitrievicMyskis (1972). Introductory mathematics for engineers: lectures in higher mathematics. Moscow: Mir.
Android Developers. (2021). Drawables overview. [online] Available at: https://developer.android.com/guide/topics/graphics/drawables#nine-patch [Accessed 16 Jul. 2022].
AviMa'ayan and Macarthur, B.D. (2012). New frontiers of network analysis in systems biology. Dordrecht; New York: Springer.
Bastian, M., Heymann, S. &Jacomy, M., (2009). Gephi: an open-source software for exploring and manipulating networks. In Third international AAAI conference on weblogs and social media.
Bisseling, R.H. (2020). Parallel scientific computation: a structured approach using BSP. Oxford, UK; New York, Ny: Oxford University Press.
Bourbaki N., "Elements of mathematics. Algebra: Algebraic structures. Linear algebra" , 1 ,Addison-Wesley (1974) pp. Chapt.1;2
Costa, R. (2020). MATRIX METHODS: applied linear algebra and sabermetrics. S.L.: Elsevier Academic Press.
Debonis, M.J. (2022). Introduction to linear algebra: computation, application, and theory. Boca Raton: C&H/Crc Press.
Faddeeva, V.N. and Benster, C.D. (1959). Computational methods of linear algebra. New York.
Frank Harold Stephenson (2016). Calculations for molecular biology and biotechnology. Amsterdam; Boston: Academic Press.
Frumusanu, A. (2014). A Closer Look at Android RunTime (ART) in Android L. [online] www.anandtech.com. Available at: https://www.anandtech.com/show/8231/a-closer-look-at-android-runtime-art-in-android-l/ [Accessed 16 Jul. 2022].
Garfinkel, A., Shevtsov, J. and Yina Guo (2017). Modeling life: the mathematics of biological systems. Cham, Switzerland: Springer International Publishing Ag.
Golub, G.H., Chan, R.H., Greif, C., O'leary, D.P. and Ebrary, I. (2007). Milestones in matrix computation: selected works of Gene H. Golub, with commentaries. New York: Oxford University Press.

Golub, G.H. and Meurant G.A. (1983). Résolutionnumérique des grands systèmeslinéaires. Paris: Editions Eyrolles.

Google Developers (2019). Meet Android Studio: Android Developers. [online] Android Developers. Available at: https://developer.android.com/studio/intro.

IEEE Computer Society, IEEE Circuits and Systems Society and IEEE Electron Devices Society (1993). Proceedings: 1993 IEEE International Conference on Computer Design: VLSI in computers & processors. Los Alamitos, Calif.: IEEE Computer Society Press; Piscataway, Nj.

Ismail, A., Ibrahim, M.S., Ismail, S., Aziz, A.A., Yusoff, H.M., Mokhtar, M. and Juahir, H. (2022). Development of COVID-19 Health-Risk Assessment and Self-Evaluation (CHaSe): a health screening system for university students and staff during the movement control order (MCO). Network Modeling Analysis in Health Informatics and Bioinformatics, 11(1). doi:10.1007/s13721-022-00357-3.

Kaw, A.K. (2008). Matrix algebra. Tampa, Fl.

Kolaczyk, E.D. (2010). Statistical analysis of network data: methods and models. New York: Springer.

Lang S. (1966) "Linear algebra", Addison-Wesley

MeurantG.A. (1999). Computer solution of large linear systems. Amsterdam; New York: North-Holland.

Khanna, V.V., Chadaga, K., Sampathila, N., Prabhu, S., Chadaga, R. and Umakanth, S. (2022). Diagnosing COVID-19 using artificial intelligence: a comprehensive review. Network Modeling Analysis in Health Informatics and Bioinformatics, 11(1). doi:10.1007/s13721-022-00367-1.

Proskuryakov, I.V. (2011). Cramer rule - Encyclopedia of Mathematics. [online] encyclopediaofmath.org. Available at: http://encyclopediaofmath.org/index.php?title=Cramer_rule&oldid=14865 [Accessed 15 Jul. 2022].

Samsung levant. (n.d.). Samsung A30 - Phone Overview. [online] Available at: https://www.samsung.com/levant/smartphones/galaxy-a-series/a30/.

Sewell, G. (2017). Computational methods of linear algebra. New Jersey Etc.: World Scientific. C.

Shores, T.S. (2018). Applied linear algebra and matrix analysis. Cham, Switzerland: Springer.

Slepian, P. (1968). Mathematical foundations of network analysis. Berlin; New York Etc.: Springer.

Stanimirovi, I. (2017). Computation of Generalized Matrix Inverses and Applications. Apple Academic Press.

Sundaram Seshu and Balabanian, N. (1967). Linear network analysis. New York: Wiley.

Trefethen, Lloyd N.; Bau, David (1997), Numerical linear algebra, Philadelphia: Society for Industrial and Applied Mathematics, ISBN 978-0-89871-361-9

www.guardsquare.com. (n.d.). ProGuard Manual: Upgrading | Guardsquare. [online] Available at: https://www.guardsquare.com/manual/setup/upgrading [Accessed 16 Jul. 2022].

www.wolframalpha.com. (1996). lu Decomposition – Wolfram |Alpha. [online] Available at: https://www.wolframalpha.com/input?i=lu+Decomposition+ [Accessed 16 Jul. 2022].

Xian-Da Zhang (2017). Matrix analysis and applications. Cambridge New York, Ny Port Melbourne Cambridge University Press.

Yang, M.-H., Yang, F.-Y. and Oyang, Y.-J. (2013). Application of density estimation algorithms in analyzing co-morbidities of migraine. Network Modeling Analysis in Health Informatics and Bioinformatics, 2(2), pp.95–107. doi:10.1007/s13721-013-0028-8.

Yang, W., Zhang, D., Peng, L., Zhuge, C. and Hong, L. (2021). Rational evaluation of various epidemic models based on the COVID-19 data of China. arXiv:2003.05666 [math, q-bio]. [online] Available at: https://arxiv.org/abs/2003.05666v2 [Accessed 16 Jul. 2022].

Young, D.M. (2003). Iterative solution of large linear systems. Mineola, N.Y.: Dover Publications.